Lean Verification Techniques: Executable SystemVerilog UVM Defect Table For Simulations

Kamel Belhous, Paul Ulrich: Teradyne, Boston MA

Steve Burchfiel, Kevin Schott: CorrectDesigns, Austin TX

accellera SYSTEMS INITIATIVE

2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Agenda

- Overview
- Use Model Examples
- How Is It Built?
- Audits
- Future Enhancements

Introduction

- How do verification engineers ….
  - Keep the simulation environment stable?
  - Prevent regression failures on features that once worked?
  - Maintain passing regressions to assess the quality of the design?
  - Release a chip knowing all work arounds have been removed?

  All while new RTL, checkers, and tests are being added and debugged.

- This presentation introduces an 'executable' defect table
  - To track and work around open issues
  - For the design and verification environments
  - The existing regression suites can continue to be used to qualify changes, and find new, unknown flaws in the code base
  - Verification engineers can make progress, without being slowed down by the existing bugs which have not been fixed yet by the design teams
- A lean verification technique used to reduce the waste/waiting/delays, by providing the ability to mitigate productivity losses, due to the impact of the delays introduced by design, specification, and verification changes for bug fixes

The Defect Table

- The defect table is SystemVerilog code that is built into the verification environment
  - Each entry tracks the state of an issue.
  - The defect's state can affect behavior of the testbench (e.g. checker, work-around).

- The defect table provides
  - A disciplined approach to using a designated class to keep track of work-arounds in the testbench when bugs exist in the design, and/or new unsupported features have been added that might otherwise cause regressions to fail.
  - A consistent de-facto location where "TODO"s are placed that affect the testbench's behavior to provide run time decisions on whether a work-around should be applied or not.
  - Designers and DV engineers a stable model, and an easy way to verify fixes.
  - DV engineers a way to release support for future RTL features; enabling the designer to test changes and update the defect table prior to release.
  - A means to review the complete list of work-arounds and TODOs to assure the proper work-arounds have been applied or removed prior to tape-out.
  - More useful when the number of bugs is large, or when the number of unresolved bugs have "piled up".

- Notes
  - The entire suite of tests, checkers, and coverage continue to be released while the designers debug and fix the problems.
  - The defect table is not a replacement for the existing official bug tracking systems.

Use Models

- Demote scoreboard errors to warnings
- Disable a check
- Constrain randomization, based on the state of a defect to avoid certain types of stimulus

- Examples…..

Controlling Error Messages

- Register should not be set it ticket is closed: ADC allowed to be BUSY only when the defect is not in the CLOSED state

```
// Make sure a record mode measurement is not active
if (reg_block.chip_adc_ctrl_reg.busy.get()==1) begin
  if(defect_settings_c::get_state_for_defect("ADC_SERIAL_DBG") inside {CQ_CLOSED}) begin
    `uvm_fatal(get_name(), $sformatf("ADC RECORD_MODE FSM is unexpectedly busy, chip_adc_mode_reg=%s", reg_block.chip_adc_mode_reg.convert2string()) )
  end
end
```

- Disable scoreboards via UVM config_db: address and data errors will be demoted to warnings, until a specific bug is fixed

```
if (defect_is_state("JASIC00051131" , {CQ_NEW})) begin
  uvm_config_db#(uvm_bitstream_t)::set(get_parent(), "*.chip_sat_model_spi_cfg_*_final_addr_scb" , "demote_errors_to_warnings", 1);
  uvm_config_db#(uvm_bitstream_t)::set(get_parent(), "*.chip_sat_model_spi_cfg_*_final_data_scb" , "demote_errors_to_warnings", 1);
end
```

Controlling Constraints

- If a defect is open, constrain the driver to use DRV_TIMING_ORIG timing, not DRV_TIMING_MIN timing. The verification environment will not create traffic using DRV_TIMING_MIN until the defect is closed/fixed

```
class chip_adc_driver_c extends uvm_driver#(chip_adc_trans_c);
  `uvm_component_utils(chip_adc_driver_c)

  typedef enum {DRV_TIMING_ORIG, DRV_TIMING_MIN} adc_driver_timing_e_t;

  rand adc_driver_timing_e_t adc_driver_timing_e;

  constraint adc_driver_timing_e_ct {
    (defect_settings_c::get_state_for_defect("JASIC00049909") inside {CQ_NEW, CQ_OPENED}) -> adc_driver_timing_e == DRV_TIMING_ORIG;
```

Static function in defect package

Constrain randomization

Simulation Output (Log File)

- List of actively tracked defects

```
UVM_INFO @  0.000ns uvm_test_top.chip_env.m_chip_defects >tvm_defect_utils_pkg.sv(120) [m_chip_defects] Current defects:

DV Issues (arbitrary label)
  id=PSET_MEM_MODELING    CQ_NEW       PERGEN    PSET host access while pattern running requires modeling pset memory
  id=LVLS_REGS_SCBS       CQ_NEW       LEVELS    Debugging levels reference model
  id=RELAY_TIMER          CQ_NEW       RELAYS    Turn off checker until reset treatment is correct

RTL Bugs (real tickets)
  id=JASIC00048949        CQ_NEW       ADC       ADC Record Mode does not write to memory.  Wdata showing 0's while record mode is working
  id=JASIC00049514        CQ_NEW       TMU       TMU_TIMESTAMP_CNT is not reset when TMU rearms itself at pattern mode
  id=JASIC00049714        CQ_NEW       PCIE      Reset values for DMA registers need to be documented
  id=JASIC00048322        CQ_CLOSED    CHANNEL   Receive pipeline is 2 clk400 cycles shorter than the drive pipepline
```

- Workarounds being applied

```
UVM_INFO @  0.000ns chip_defect_settings_c.sv(530) [m_chip_defects] Applying work-around to tb code to account for PSET_MEM_MODELING
UVM_INFO @  0.000ns chip_defect_settings_c.sv(573) [m_chip_defects] Applying work-around to tb code to account for LVLS_REGS_SCBS
UVM_INFO @  0.000ns chip_defect_settings_c.sv(578) [m_chip_defects] Applying work-around to tb code to account for RELAY_TIMER
UVM_INFO @  0.000ns chip_defect_settings_c.sv(595) [m_chip_defects] Applying work-around to tb code to account for JASIC00048949
UVM_INFO @  0.000ns chip_defect_settings_c.sv(599) [m_chip_defects] Applying work-around to tb code to account for JASIC00049514
UVM_INFO @  0.000ns chip_defect_settings_c.sv(599) [m_chip_defects] Applying work-around to tb code to account for JASIC00049714
UVM_INFO @  0.000ns chip_defect_settings_c.sv(639) [m_chip_defects] Demoting errors to account for JASIC00049714
```

How Is It Built?

- The common base source code for a defect table is project and block independent such that a defect table can be reused and integrated into multiple environments if desired
- Base classes provide the API for defining and accessing a list of defects (common across all projects)
- Projects just implement a project “defect_settings” class, which contains the exact set of defects for a given block or project
- User code can query the state of a defect to take specific actions
- Users can also put their defect dependent actions directly in the defect_settings file (e.g. demote an error to a warning for a specific checker), which is the recommended way for creating an easy, common location for auditing the workaround behavior.

Defect Info Base Class

- The main purpose of this class is to capture information about the defects (subset of content in the real bug tracking system) and provide an API for other components in the system to get information about the defects.
- Contains field values for each defect along with accessor methods.
- Project independent

```
typedef enum {CQ_NEW, CQ_OPENED, CQ_RESOLVED, CQ_CLOSED, CQ_UNDEFINED} defect_state_e_t;

class defect_info_base_c extends uvm_component;
  `uvm_component_utils(defect_info_base_c)

  defect_state_e_t  state_e;
  string            id;
  string            summary;

  function new(string name="defect_info_base_c", uvm_component parent=null);

  virtual function void             create_defect(string id, string features, string summary);

  virtual function void             set_state(defect_state_e_t state_e);
  virtual function defect_state_e_t get_state();

  virtual function void             set_features(string features);
  virtual function string           get_features();

endclass
```

```
class defect_settings_c extends uvm_component;
  `uvm_component_utils(defect_settings_c)

  static defect_info_base_c defect_id_q[$];

  function new(string name="defect_settings_c", uvm_component parent=null);

  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    define_defects();
  endfunction

  virtual function void end_of_elaboration_phase(uvm_phase phase);
    super.end_of_elaboration_phase(phase);
  endfunction

  virtual function void create_defect(string id, defect_state_e_t state, string features, string description);
    defect_info_base_c defect = defect_info_base_c::type_id::create(id, this);
    defect.create_defect(id, features, description);
    defect.set_state(state);
    defect_id_q.push_back(defect);
  endfunction

  virtual function void define_defects();
  endfunction

  static function automatic bit defect_is_state(string id, defect_state_e_t state_q[$]);
    int idx_q[$] = defect_id_q.find_index(d) with (d.id==id && d.state_e inside {state_q});
    return (idx_q.size()!=0);
  endfunction

  static function automatic defect_state_e_t get_state_for_defect(string id);
    defect_info_base_c defect_q[$] = defect_id_q.find_first(d) with (d.id==id);
    if (defect_q.size()!=0) begin
      return defect_q[0].state_e;
    end else begin
      return CQ_UNDEFINED;
    end
  endfunction

endclass
```

Queue of project specific defects

UVM Phased Component

Project specific: calls to create_defect()

User code: take action

Project Specific Defect List & Actions

```
class chip_defect_settings_c extends defect_settings_c;
  `uvm_component_utils(chip_defect_settings_c)

  typedef enum {ADC, XADC, LEVELS, PCIE, AXI, DDR, RELAY, NVM } chip_features_e_t;

  function new(string name="chip_defect_settings", uvm_component parent=null);
    super.new(name, parent);
  endfunction

  virtual function void build_phase(uvm_phase phase);
    set_type_override_by_type(.original_type(defect_info_base_c::get_type()), .override_type(defect_info_c#(chip_features_e_t)::get_type()));
    super.build_phase(phase);
  endfunction

  //--------------------------------

  virtual function void define_defects();
    super.define_defects();

    create_defect("PSET_MEM_MODELING", CQ_NEW    , "PERGEN"       , "PSET host access while pattern running requires modeling pset memory");
    create_defect("LVLS_REGS_SCBS" , CQ_NEW      , "LEVELS"       , "Debugging levels reference model.");
    create_defect("RELAY_TIMER"    , CQ_NEW      , "TESTBENCH"    , "Turn off checker until reset treatment is correct");

    create_defect("JASIC00048949"  , CQ_NEW      , "ADC"          , "ADC Record Mode does not write to memory.");
    create_defect("JASIC00049514"  , CQ_CLOSED   , "TMU"          , "TMU_TIMESTAMP_CNT is not reset when TMU rearms itself");
    create_defect("JASIC00049714"  , CQ_NEW      , "PCIE"         , "Reset values for DMA registers need to be documented");
    create_defect("JASIC00048322"  , CQ_CLOSED   , "CHANNEL"      , "RCV pipeline is 2 clk cycles shorter than the DRV pipepline");
  endfunction
  virtual function void adjust_tb_for_defects();
    super.adjust_tb_for_defects();

    if (defect_is_state("chip_SEQ_FAIL_CNT" , {CQ_NEW})) begin
      uvm_config_db#(uvm_bitstream_t)::set(get_parent(), "*.seq_fail_cnt_scb" , "demote_errors_to_warnings", 1);
    end

    ...

  endfunction

endclass
```

Define the project specific defects
(Suggestion: organize table by owner to avoid merge conflicts if multiple
engineers need to edit this file).

Good common location to take action based on defect state. Called during end_of_elaboration()

"JASIC000xxxx" are four defects which are also tracked in the official bug tracking system. The state(CQ_NEW/CQ_CLOSED) indicates whether the issue is considered resolved in this specific build of the environment and will impact how the verification environment behaves.

The three text entries (PSET_MEM_MODELING, LVLS_REGS_SCBS, RELAY_TIMER) represent DV environment issues/bugs. They are not officially tracked in the bug tracking system. This gives DV engineers an easy way to alter the behavior of the verification environment, without having to track every change with an official ticket providing an easy, low-cost way, but trackable way to release new verification code that is not ready for 100% production/regression use. If an issue does not affect how the verification environment behaves, it should only be tracked in the official bug tracking system.

Constructing The Defect Table(s)

- The defect tables are UVM Components:
  - Just create in uvm_env::build()
  - Defect tables from IP blocks, or lower level modules can also be built

```
class chip_env_c extends uvm_env;

  `uvm_component_utils(chip_env_c)

  defect_settings_c m_chip_defects;
  defect_settings_c m_seq_defects;

  virtual function void build_phase(uvm_phase phase);
    m_chip_defects    = chip_defect_settings_c::type_id::create("m_chip_defects",this);
    m_seq_defects     = seq_defect_settings_c: :type_id::create("m_seq_defects",this);
    super.build_phase(phase);
  endfunction
```

Two defect tables in this env

Ready for Tape-Out?

- Automated report compares defect_settings.sv with real bug tracking system

```
Defect table report generated on 2016.08.05 via /u/burchfs/bin/defect_table_report.sh
  Open:       5   // Marked as !CQ_CLOSED in the defect_table
  Closed:     2   // Marked as  CQ_CLOSED in the defect_table
  Total:      7   // Total number of entries in the defect_table
  CanClose:   1   // According to the actual state in ClearQuest, see the end of this file for the list.

  Defect settings table:
    create_defect("PSET_MEM_MODEL" , CQ_NEW      , "PERGEN"    , "PSET host access while pat running");
    create_defect("LVLS_REGS_SCBS" , CQ_NEW      , "LEVELS"    , "Debugging levels reference model.");
    create_defect("RELAY_TIMER"    , CQ_NEW      , "TESTBENCH" , "Turn off checker until reset treatment is correct");
    create_defect("JASIC00048949"  , CQ_NEW      , "ADC"       , "ADC Record Mode does not write to memory.");
    create_defect("JASIC00049714"  , CQ_NEW      , "PCIE"      , "Reset values for DMA registers need to be documented");
    create_defect("JASIC00049514"  , CQ_CLOSED   , "TMU"       , "TMU_TIMESTAMP_CNT is not reset when TMU rearms itself");
    create_defect("JASIC00048322"  , CQ_CLOSED   , "CHANNEL"   , "RCV pipeline is 2 clk cycles shorter than the DRV pipepline");

 Defects closed in the actual CQ system thar are marked CQ_NEW in the defect_table...
    Closed  JASIC00048322   Design RTL      RtlBug     Kamel  "RCV pipeline is 2 clk cycles shorter than the DRV pipepline"
```

- Review TODO's, FIXMEs & code comments

  OR

  defect_settings.sv and the simulation log files?

Benefits Review

- Regressions only fail because of new bugs
  - Tests are kept active but specific checks/randomization is limited
  - DV engineers are “less” slowed down by the existing bugs which have not been fixed yet by the design teams
- Simulation logfile indicates which defects affect the env
- Automation easily indicates when an open defect table entry should be closed
- Enable the design engineers to test changes and update the defect table prior to releasing the code
- Provides the ability to thoroughly review the complete list of workarounds and TODOs (they are supported in a common manner) to ensure the proper workarounds have been applied and removed prior to tape-out
- Single location to audit work arounds. No need to worry about:
  - Commented out code, and forgotten changes that limit verification

Future Enhancements

- Command line defect overrides
  - To verify RTL fixes without modifying the defect_settings.sv file
  - To enable running tests for reproducing failures or testing new RTL features without code modification
- System call to a live defect query
  - Automatically compare defect table entries to the actual bug management tool
  - Automatically run tests with fixed issues to check correctness
- Apply other lean techniques to DV
  - Value Stream Map: flowchart to manage/analyze/improve dependencies and handoffs, to identify bottlenecks, reveal opportunities for automation,..
  - Andons: automatic email notification for bugs with longer turnaround times

Questions

Thank You !